Iterated Greedy Algorithm para o problema da Distribuição Ótima em GPU's

Vítor Hugo Magnus Oliveira   Marcelo Gross Borges

Janeiro 2025

1 Introdução:

Esse trabalho teve como objetivo desenvolver, implementar e testar uma meta-heurística do tipo Iterated Greedy Algorithm (IGA) para resolver o problema da Distribuição Ótima em GPUs. Nesse problema temos n GPU's todas com a mesma quantidade de VRAM, e m partes de uma rede neural (PRN), com um consumo de VRAM e um tipo. A solução ideal envolve reduzir o máximo possível a soma de tipos de PRN por GPU para todas GPU's.

2 Formulação Matemática:

2.1 Parâmetros

- $n \in \mathbb{Z}_+$: Número de GPUs.
- $m \in \mathbb{Z}_+$: Número de partes de uma rede neural (PRN).
- $V \in \mathbb{R}_+$: Capacidade máxima de VRAM de cada GPU em GB (todas GPUs tem o mesmo limite).
- $v_j \in \mathbb{R}_+$: Armazenamento necessário para cada PRN j em GB.
  - $j \in [m]$.
- T: Conjunto dos possíveis tipos de PRN.
- $t_j \in T$: Tipo da PRN j.
  - $j \in [m]$.

2.2 Variáveis de Decisão:

- $x_{i,j} = \begin{cases} 1, & \text{se a GPU } i \text{ está processando a PRN } j, \\ 0, & \text{caso contrário.} \end{cases}$
  - $i \in [n]$.
  - $j \in [m]$.
- $y_{i,j} = \begin{cases} 1, & \text{se a GPU } i \text{ está processando uma PRN do tipo } j, \\ 0, & \text{caso contrário.} \end{cases}$
  - $i \in [n]$.
  - $j \in T$.

2.3 Formulação:

$$
\begin{aligned}
\textbf{minimiza} \quad & \sum_{\substack{i \in [n] \\ j \in [T]}} y_{i,j} & & \\
\textbf{sujeito a} \quad & \sum_{j \in [m]} x_{i,j} v_j \leq V & \forall i \in [n], & \quad (1) \\
& \sum_{i \in [n]} x_{i,j} = 1 & \forall j \in [m] & \quad (2) \\
& x_{i,j} \leq y_{i,t_j} & \forall i \in [n], \forall j \in [m] & \quad (3) \\
& y_{i,j} \in \{0,1\} & \forall i \in [n], \forall j \in T, & \quad (4) \\
& x_{i,j} \in \{0,1\} & \forall i \in [n], \forall j \in [m], & \quad (5)
\end{aligned}
$$

2.4 Descrição das restrições:

(1) Garante que o total de memória ocupada por PRNs em cada GPU não ultrapasse o limite máximo de VRAM.

(2) Assegura que cada parte da rede neural (PRN) seja processada exatamente em uma única GPU.

(3) Vincula a alocação de tipos de PRNs às GPUs específicas que as processam.

(4) Limita o número de GPUs distintas que podem processar cada tipo de rede neural.

(5) O número de GPUs distintas utilizadas deve ser um número inteiro maior ou igual a zero.

(6) As variáveis de tipo de GPU são valores binários (0 ou 1).

(7) As variáveis de alocação de PRNs são valores binários (0 ou 1).

3 Algoritmo

Iterated Greedy Algorithm é uma meta-heurística que opera por meio da iteração de duas etapas fundamentais: destruição e reconstrução. Durante a fase de destruição o algoritmo destrói parcialmente uma solução candidata completa. Em seguida, na fase de construção o algoritmo utiliza técnicas gulosas para reconstruir uma solução canditada completa potencialmente melhor que a anterior.

3.1 Representação da Solução

A solução será uma matriz de inteiros com $|m|$ linhas e 4 colunas, onde m representa o conjunto de PRNs (Processos de Recurso Necessário). Cada linha corresponde a uma alocação específica, com as seguintes informações:

- **Coluna 1**: Índice único do PRN no conjunto m.
- **Coluna 2**: Capacidade requerida pelo PRN.
- **Coluna 3**: Tipo ou categoria do PRN.
- **Coluna 4**: Índice da GPU alocada ao PRN, pertencente ao conjunto n, que representa as GPUs disponíveis.

Formalmente, $p_{1,j} \in m$ identifica o índice do PRN, $p_{2,j}$ representa a capacidade requerida, $p_{3,j}$ define o tipo do PRN, e $p_{4,j} \in n$ indica o índice da GPU alocada, onde n é o conjunto de GPUs disponíveis.

3.2 Definição Formal das variáveis

Formalmente, considere o conjunto de GPUs $= \{gpu_1, gpu_2, \dots, gpu_m\}$, onde cada gpu_i possui uma capacidade máxima de VRAM $C(gpu_i)$, uma capacidade de VRAM ocupada $O(gpu_i)$ e um conjunto de PRNs $P(gpu_i)$ o conjunto de PRNs $= \{prn_1, prn_2, \dots, prn_n\}$, onde cada prn_j requer uma quantidade de VRAM $R(prn_j)$ e possui um tipo $T(prn_j)$.

3.3 Solução Inicial

A solução inicial é gerada de maneira sequencial utilizando uma abordagem gulosa para alocar cada PRN no primeiro recurso disponível que satisfaça as restrições de capacidade de memória. Essa abordagem garante que todos os $p_j \in P$ sejam alocados em GPUs sem exceder a capacidade de VRAM. Embora a solução inicial seja válida, ela não necessariamente minimiza o número total de GPUs utilizadas, devido à sua natureza gulosa e sequencial. A construção da solução inicial segue o Algorithm 1.

3.4 Destruição e Construção

Durante a fase de destruição do algoritmo, selecionam-se aleatoriamente três GPUs do conjunto disponível para serem desmontadas. Em seguida, na fase de construção, as PRNs associadas a essas três GPUs são ordenadas por tipo e realocadas de forma gulosa, resultando na criação de novas GPUs. Esse procedimento visa melhorar a função objetivo ao agrupar PRNs do mesmo tipo em GPUs iguais. A função de destruição está definida no Algorithm 3 e a função de construção está definida no Algorithm 4.

Algorithm 1 Solução Inicial Gulosa

Require: Conjunto de PRNs ordenados por tipo, conjunto inicial de GPUs vazio
Ensure: Alocação válida de PRNs em GPUs respeitando restrições de VRAM

for cada $prn_j \in PRNs$ **do**
  $gpu\_index \leftarrow 0$
  while $O(g_{gpu\_index}) + R(prn_j) > C(g_{gpu\_index})$ **do**
    $gpu\_index \leftarrow gpu\_index + 1$
    if $gpu\_index = |GPUs|$ **then**
      Criar nova gpu_{m+1} com $O(gpu_{m+1}) \leftarrow 0$ e $P(gpu_{m+1}) \leftarrow \emptyset$
      $GPUs \leftarrow GPUs \cup \{gpu_{m+1}\}$
    end if
  end while
  $P(g_{gpu\_index}) \leftarrow P(g_{gpu\_index}) \cup \{prn_j\}$
  $O(g_{gpu\_index}) \leftarrow O(g_{gpu\_index}) + R(prn_j)$
end for
return GPUs =0

3.5 Critério de Parada

O critério de parada é o número de iterações do algoritmo. Ao atingir um número máximo de iterações o algoritmo é encerrado.

3.6 Temperatura

A temperatura é um parâmetro que controla a aceitação de novas soluções e é constante durante toda a execução do algoritmo na nossa implementação. Uma temperatura alta permite aceitar mais soluções piores, favorecendo exploração, enquanto uma temperatura baixa torna o algoritmo mais seletivo, aceitando apenas soluções que melhoram a solução atual. O critério de aceitação é aplicado sempre que uma nova solução é construída após a fase de destruição e reconstrução, permitindo que o algoritmo ocasionalmente aceite soluções piores para escapar de ótimos locais, mas mantendo uma tendência geral de convergência para soluções melhores.

Algorithm 2 Iterated Greedy Algorithm

procedure ITERATEDGREEDY($max\_iterations, temperature$)
  $current\_solution\_global \leftarrow$ AvaliateSolucao($GPUs$)
  $best\_solution \leftarrow current\_solution\_global$
  $best\_gpus \leftarrow GPUs$
  for $i \leftarrow 1$ to $max\_iterations$ **do**
    $gpus \leftarrow GPUs$.copy()
    $(prns, current\_solution) \leftarrow$ Destroy($gpus$)
    $(new\_solution, gpus) \leftarrow$ Construct($prns, gpus$)
    $delta \leftarrow new\_solution - current\_solution$
    if $|gpus| \leq gpu\_n$ **and** AcceptSolution($delta, temperature$) **then**
      $current\_solution\_global \leftarrow current\_solution\_global - current\_solution + new\_solution$
      $GPUs \leftarrow gpus$
      if $current\_solution\_global < best\_solution$ **then**
        $best\_solution \leftarrow current\_solution\_global$
        $best\_gpus \leftarrow gpus$
      end if
    end if
  end for
  $GPUs \leftarrow best\_gpus$
  return $best\_solution$
end procedure=0

Algorithm 3 Destroy Function

procedure DESTROY($gpus$)
  $available\_gpus \leftarrow gpus$.copy()
  $gpu1 \leftarrow$ RandomChoice($available\_gpus$)
  Remove($available\_gpus, gpu1$)
  $gpu2 \leftarrow$ RandomChoice($available\_gpus$)
  Remove($available\_gpus, gpu2$)
  $gpu3 \leftarrow$ RandomChoice($available\_gpus$)
  $current\_solution \leftarrow$ AvaliateSolucao($[gpu1, gpu2, gpu3]$)
  $prns \leftarrow P(gpu1) \cup P(gpu2) \cup P(gpu3)$
  Remove($gpus, \{gpu1, gpu2, gpu3\}$)
  return $(prns, current\_solution)$
end procedure=0

Algorithm 4 Construct Function

procedure CONSTRUCT($prns, gpus$)
  Ordenar($prns$) por tipo
  $new\_gpus \leftarrow [\{P = \emptyset, O = 0\}]$
  for $prn_j \in prns$ **do**
    $gpu\_index \leftarrow 0$
    while $O(g_{gpu\_index}) + R(prn_j) > C(g_{gpu\_index})$ **do**
      $gpu\_index \leftarrow gpu\_index + 1$
      if $gpu\_index \geq |new\_gpus|$ **then**
        new_gpus.append($\{P = \emptyset, O = 0\}$)
      end if
    end while
    $P(g_{gpu\_index}) \leftarrow P(g_{gpu\_index}) \cup \{prn_j\}$
    $O(g_{gpu\_index}) \leftarrow O(g_{gpu\_index}) + R(prn_j)$
  end for
  $new\_value \leftarrow$ AvaliateSolucao(new_gpus)
  $gpus \leftarrow gpus \cup new\_gpus$
  return $(new\_value, gpus)$
end procedure=0

Algorithm 5 Accept Solution Function

procedure ACCEPTSOLUTION($delta, temperature$)
  $probability \leftarrow e^{-delta/temperature}$
  return Random$(0, 1) < min(1, probability)$
end procedure=0

Algorithm 6 Evaluate Solution Function

procedure EVALUATESOLUTION($gpus$)
  function GPUTYPEDISTRIBUTION(gpu)
    $types \leftarrow \emptyset$ {Empty set to store unique types}
    for all $prn\_index \in P(gpu)$ **do**
      if $Type(prn\_index) \notin types$ **then**
        $types \leftarrow types \cup \{Type(prn\_index)\}$
      end if
    end for
    return $|types|$ {Return number of unique types}
  end function
  $value \leftarrow 0$
  for all $gpu \in gpus$ **do**
    $value \leftarrow value+$ GPUTYPEDISTRIBUTION(gpu)
  end for
  return $value$
end procedure=0

4 Implementação

4.1 Estruturas de Dados Utilizadas

Durante a implementação, a solução foi modelada como uma lista de GPUs, onde cada GPU foi representada por uma estrutura contendo dois campos: um inteiro que indica a quantidade de VRAM ocupada e uma lista de inteiros representando os índices das PRNs alocadas. Por sua vez, cada PRN foi definida como uma estrutura com dois atributos: um inteiro que especifica a quantidade de VRAM requerida e uma string que identifica o tipo da PRN.

5 Teste de Parâmetro

5.1 Temperatura

Com o parâmetro de temperatura, realizamos experimentos com quatro valores distintos: 0.01, 0.5, 1.0 e 5.0, aplicados em todas as instâncias do problema. Estes valores foram escolhidos para cobrir diferentes níveis de aceitação de soluções, desde uma abordagem mais conservadora até uma mais exploratória.

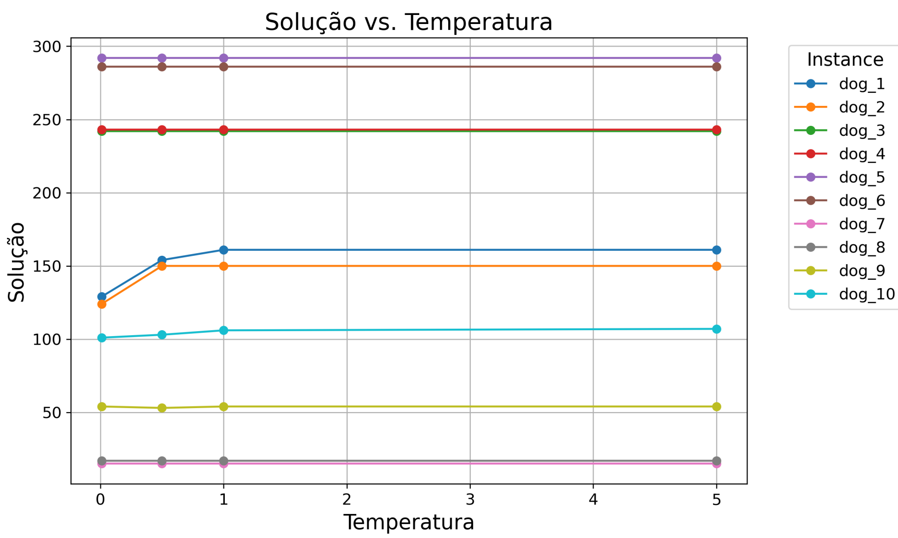

Figure 1: Solução vs. Temperatura

Analisando os resultados apresentados em Figure 1, é possível perceber que temperaturas menores geraram consistentemente melhores resultados. Portanto, os testes das instâncias foram realizados com a temperatura mais baixa, 0.01.

6 Teste das Instâncias

Os valores de BKV apresentados na Tabela 1 são dados fornecidos pelo professor que representam o Best Known Value (melhor valor conhecido da instância). Esses valores serão utilizados como parâmetro para comparar com os resultados obtidos.

Table 1: Valores referentes a cada instância

Instâncias	BKV
dog01	128
dog02	123
dog03	238
dog04	238
dog05	285
dog06	279
dog07	15
dog08	16
dog09	51
dog10	118

6.1 Iterated Greedy

O algoritmo Iterated Greedy, descrito no Algorithm 2, foi executado com uma temperatura de 0.01, um limite de 1 milhão de iterações e uma seed igual 42.

Table 2: Comparação BKV e Solução Inicial com Iterated Greedy para cada instância

Instâncias	Valor IG	BKV	Solução Inicial	Desvio BKV	Desvio SI	Tempo de Execução
dog01	129	128	161	0.78%	-19.88%	30.47s
dog02	124	123	150	0.81%	-17.33%	30.85s
dog03	242	238	242	1.68%	0.00%	47.18s
dog04	243	238	243	2.10%	0.00%	47.57s
dog05	292	285	292	2.46%	0.00%	53.97s
dog06	286	279	286	2.51%	0.00%	52.99s
dog07	15	15	16	0.00%	-6.25%	53.98s
dog08	17	16	18	6.25%	-5.56%	57.02s
dog09	54	51	54	5.88%	0.00%	30.59s
dog10	101	118	107	-14.41%	-5.61%	37.49s

Analisando os resultados da Tabela 2, observa-se que, para as instâncias dog03, dog04, dog05, dog06 e dog07, não houve melhoria em relação à solução inicial. Esse comportamento ocorre pois nessas instâncias as PRNs já vem previamente ordenadas por tipo, portanto a solução inicial já é muito próxima ao BKV, fazendo com que o algoritmo tenha dificuldades de progredir.

Nas demais instâncias, como dog01 e dog02, houve melhorias significativas, indicando que a metaheurística iterated greedy foi eficaz em explorar o espaço de busca quando a solução inicial estava mais distante do BKV.

O desvio em relação ao BKV foi reduzido significativamente na instância dog10, sugerindo que o BKV estava longe da otimalidade. Nas outras instâncias, os desvios permaneceram baixos, reforçando a eficácia da combinação entre uma boa solução inicial e a metaheurística aplicada.

Por fim, o tempo de execução foi consistente e adequado para todas as instâncias, demonstrando a eficiência computacional do método proposto, mesmo em casos com maior exploração do espaço de busca.

6.2 Gurobi

Table 3: Comparação BKV com Gurobi para cada instância

Instâncias	**Valor Obtido**	**BKV**	**Desvio BKV**
dog01	118	128	-7.81%
dog02	122	123	-0.81%
dog03	374	238	57.14%
dog04	362	238	52.10%
dog05	296	285	3.86%
dog06	289	279	3.58%
dog07	14 (ótimo)	15	-6.67%
dog08	16 (ótimo)	16	0.00%
dog09	51 (ótimo)	51	0.00%
dog10	104	118	-11.86%

Gurobi é um solver de programação linear inteira mista foi utilizado para comparar o resultado da meta-heurística com técnicas tradicionais de otimização combinatória. As instâncias do problema foram passados para o solver na integra e foi definido um tempo máximo de 30 minutos para otimização do problema. Como é possível analisa na $Table 3$, em grande parte das instâncias o solver se aproximou ou superou o $BestKnownValue$, chegando à solução ótima nas instâncias dog07, dog08 e dog09. Em instâncias maiores do problema, como dog3 e dog4, o Gurobi teve resultados bem inferiores ao $BestKnownValue$, provavelmente porque o solver não foi capaz de convergir no tempo disponível.

6.3 Comparação Iterated Greedy com Gurobi

Table 4: Comparação Iterated Greedy com Gurobi para cada instância

Instâncias	Valor IG	Valor Gurobi	Desvio Gurobi
dog01	129	118	9.32%
dog02	124	122	1.64%
dog03	242	374	-35.29%
dog04	243	362	-32.87%
dog05	292	296	-1.35%
dog06	286	289	-1.04%
dog07	15	14 (ótimo)	7.14%
dog08	17	16 (ótimo)	6.25%
dog09	54	51 (ótimo)	5.88%
dog10	101	104	-2.88%

Analisando a $Table3$ é possível perceber que o algoritmo Iterated Greedy foi capaz de convergir mais fácilmente que o solver nas instâncias maiores (dog03 e dog04), e superar o Gurobi em diversas instâncias. Porém, o solver foi mais capaz para resolver problemas menores e com propensão a deixar o algoritmo Iterated Greedy preso em mínimos locais (dog07, dog08).

7 Análise dos Resultados

O Problema de Alocação de PRNs em GPUs, como uma variação do problema de Bin Packing, demonstrou-se particularmente desafiador para resolver. A implementação do Iterated Greedy Algorithm, embora capaz de encontrar soluções para todas as instâncias testadas, apresentou comportamentos distintos dependendo das características das instâncias e dos parâmetros utilizados.

7.1 Instâncias

Nossa implementação demonstrou resultados superiores em instâncias caracterizadas por número reduzido de tipos de PRNs, quantidade elevada de PRNs e espaço limitado de VRAM nas GPUs. Formulamos três hipóteses principais para explicar este comportamento: (1) poucos tipos de PRNs podem beneficiar a fase de construção ao formar grupos maiores com requisitos similares de VRAM, (2) o volume elevado de PRNs possivelmente amplifica a eficácia do mecanismo de destruição e construção ao proporcionar mais oportunidades de reorganização, e (3) a restrição de espaço de VRAM provavelmente reduz o espaço de soluções válidas, permitindo que a ordenação por tipos e a abordagem gulosa sejam mais efetivas na maximização do uso do espaço limitado.

7.2 Parâmetros

Na implementação, observamos que temperaturas mais baixas produziram consistentemente melhores resultados. Esta descoberta sugere duas hipóteses principais: (1) a natureza do problema favorece uma estratégia mais gulosa, onde a intensificação da busca em regiões promissoras é mais efetiva que a exploração ampla do espaço de soluções, ou (2) o mecanismo de destruição limitado a apenas 3 GPUs por iteração restringe significativamente a capacidade de exploração do espaço de soluções, tornando o parâmetro de temperatura menos efetivo para escapar de mínimos locais, mesmo com valores mais altos.

7.3 Conclusão

Estes resultados evidenciam a complexidade inerente ao problema, que demanda estratégias mais sofisticadas de busca local e mecanismos adaptativos de destruição/construção para alcançar soluções de melhor qualidade. Portanto, o Problema de Alocação de PRNs em GPUs continua representando um desafio significativo no campo da otimização combinatória, necessitando de abordagens mais robustas para melhorias futuras. Em particular, o desenvolvimento de mecanismos de destruição mais flexíveis e estratégias adaptativas de temperatura podem ser direções promissoras para pesquisas futuras.

8 Referências

References

[1] T. Stützle, R. Ruiz, "Iterated Greedy". In: R. Martí, P. Pardalos, M. Resende (eds) *Handbook of Heuristics*, Springer, Cham, 2018. Disponível em: https://doi.org/10.1007/978-3-319-07124-4_10.